# Optimized Selection Sort Algorithm is faster than Insertion Sort Algorithm: a Comparative Study

**Article** · April 2011

CITATIONS
19

READS
5,390

**3 authors**, including:

Muhammad Salman Faiz Solehria
Sarhad University of Science & IT
**2** PUBLICATIONS **25** CITATIONS
SEE PROFILE

Mubashir Qayyum
National University of Computer and Emerging Sciences
**44** PUBLICATIONS **321** CITATIONS
SEE PROFILE

# Optimized Selection Sort Algorithm is faster than Insertion Sort Algorithm: a Comparative Study

Sultanullah Jadoon[1], Salman Faiz Solehria[2], Mubashir Qayum[3],

*1. Department of Information Technology, Hazara University, Haripur Campus, 2. Sarhad University, Peshawar, 3. National University of Computer and Emerging Sciences, Peshawar Campus.*

sultanullah@gmail.com, ss_suit@yahoo.com, mubashir.qayyum@nu.edu.pk

**ABSTRACT –** Optimized Selection Sort Algorithm is a new sorting algorithm that has been developed to address the shortcomings of the current popular sorting algorithms. The goal of this research is to perform an extensive empirical analysis of Optimized Selection Sort against Selection Sort and Insertion Sort Algorithms. The results proved that Optimized Selection Sort is much more efficient than Selection Sort Algorithm; Furthermore analysis supports the fact that Optimized Selection Sort is better than Insertion Sort in worst case scenario.

Key Words: Optimized Selection Sort, Worst Case, Efficiency, Sorting Algorithm

## I. INTRODUCTION

SORTING algorithms are the basic foundations of practical Computer Science, and therefore, the analysis and design of useful sorting algorithms has remained one of the most important research area in the field. Despite the fact that, several new sorting algorithms being introduced, the large number of programmers in the field depends on one of the comparison-based sorting algorithms: Bubble, Insertion, Selection sort etc. The new algorithms are usually not being used as much because they are not as general purpose as comparison-based sorting algorithms, usually require more alteration to work with new classes and data types and in large cases, do not perform as well as expected due to poorer locality of reference caused by linear passes through the array [1] [2] [3] [4].

It is necessary for a new algorithm to be acknowledged and used in the field of Computer Science, the process must be shown to have as good as performance to the abovementioned sorting algorithms and be easy to use, implement, and debug. Better performance will get programmers interested in the algorithm, and ease-of-use will be the final determinate in changing the programmer's sorting preference. The easiest, most time efficient way to accomplish a task is usually preferred, and the comparison-based sorts tinted here offer excellent performance and adaptability to any type of record or data type to be sorted [5][6][7].

Insertion Sort and Selection Sort have been around for over half a century, and have been taught widely in the academic Computer Science community. These algorithms are established and well documented with literally dozens of thousands of published empirical analyses comparing and contrasting performances, highlighting theoretical boundaries, and recommending preferred solutions. This paper is no different in this aspect. However, this paper does reveal new information on a newly developed comparison-based sorting algorithm based on the Selection Sort Algorithm, known as Optimized Selection Sort Algorithm [8] [9] [15].

OSSA also referred to as Optimized Selection Sort Algorithm, is a new in place comparison-based sort. Logic of OSSA based on the old selection sort algorithm. The main difference in old SS and new OSSA is that the former sorts the data from one end i.e from largest element of array to smallest element or from smallest to largest but the later starts sorting from both ends and finds the largest and smallest data elements of array in single iteration and place them at their proper locations then during second iteration it sorts the second largest and second smallest elements from the remaining array data and place them in their appropriate location in the array. Similarly it sorts rest of the data elements and put them in their proper positions. OSSA sort the data in half iterations as compared to old SS. It sorts two data elements at a time, which is why it minimizes the sorting time up to 50%. Other characteristics of this algorithm are as under [15].

- It is an in place algorithm.
- It is simple.
- Although it is O ($n^2$) Algorithm, yet it is faster than all algorithms of this order.

- It is two times faster than old SS and Almost as
- It is a Stable algorithm.
- It performs less data exchange operation than bubble or insertion sort algorithm of this order.

## II. SELECTION SORT ALGORITHM

### *A. Selection Sort*

It is one of the easiest and most useful sorting algorithms for dealing with small amount of data set. Even though it performs a lot of comparisons, but it performs the least amount of data moving. Thus, if your data has small keys but large data area, then selection sorting may be the quickest [15].

### *B. Pseudo Code and Execution Time of Individual Statement of Old Selection Sort*

**Algorithm SelectionSort (X, n)** ▷ **X[0..n-1]**

| S. No | Iteration | Cost | Times |
|---|---|---|---|
| 1 | **FOR** i ← n – 1 **TO** 0 | $C_1$ | $n$ |
| 2 | IndexOfLarge ← 0 | $C_2$ | $n-1$ |
| 3 | **FOR** j ← 1 **TO** n | $C_3$ | $\sum_{i=1}^{n-1}(i+1)$ |
| 4 | if (X[j]>X[IndexOfLarge)) | $C_4$ | $\sum_{i=1}^{n-1}(i)$ |
| 5 | IndexOfLarge ← j | $C_5$ | $\sum_{j=2}^{n} i(t_j)$ |
| 6 | Large ←X[IndexOfLarge] | $C_6$ | $n-1$ |
| 7 | X[IndexOfLarge] ← X[i] | $C_7$ | $n-1$ |
| 8 | X[i] ← Large | $C_8$ | $n-1$ |

### *C. Best-Case Time Complexity of Old Selection Sort.*

For the best-case scenario when we have all $t_{ij} = 0$

$$T(n) = C_1 n + C_2(n-1) + C_3\sum_{i=1}^{n-1}(i+1) + C_4\sum_{i=1}^{n-1}(i) + C_5\sum_{i=1}^{n-1}(i) + C_6(n-1) + C_7(n-1) + C_8(n-1)$$

$$C_3\sum_{i=1}^{n-1}(i+1) = C_3\frac{n(n-1)}{2} + C_3(n-1)$$

$$C_4\sum_{i=1}^{n-1}(i) = C_4\frac{n(n-1)}{2}$$

$$C_5\sum_{i=1}^{n-1}(i+1) = C_5\frac{n(n-1)}{2}$$

$$T(n) = C_1 n + C_2(n-1) + C_3\frac{n(n-1)}{2} + C_3(n-1) + C_4\frac{n(n-1)}{2} + C_5\frac{n(n-1)}{2} + C_6(n-1) + C_7(n-1) + C_8(n-1)$$

$$T(n) = C_1 n + C_2 n - C_2 + C_3\frac{n(n-1)}{2} + C_3(n-1) + C_4\frac{n(n-1)}{2} + C_6 n - C_6 + C_7 n - C_7 + C_8 n - C_8$$

$$T(n) = C_1 n + C_2 n - C_2 + C_3 n^2/2 - C_3 n/2 + C_3 n - C_3 + C_4 n^2/2 - C_4 n/2 + C_6 n - C_6 + C_7 n - C_7 + C_8 n - C_8$$

$$T(n) = \left(\frac{C_3}{2} + \frac{C_4}{2}\right)n^2 + \left(C_1 + C_2 + C_3 + C_6 + C_7 + C_8 - \frac{C_3}{2} - \frac{C_4}{2}\right)n - (C_2 + C_3 + C_7 + C_8)$$

Let

$$a = \frac{C_3}{2} + \frac{C_4}{2}$$

$$b = C_1 + C_2 + C_3 + C_6 + C_7 + C_8 - \frac{C_3}{2} - \frac{C_4}{2}$$

$$c = -(C_2 + C_3 + C_7 + C_8)$$

Then replacing the values of $a, b$ and $c$ in $T(n)$ we have

$$T(n) = an^2 + bn + c$$

Thus here in best-case, the complexity of execution time of an algorithm shows the lower bound and is asymptotically denoted with Ω. Therefore by ignoring the constant a, b, c and the lower terms of n, and taking only the dominant term i.e. $n^2$, then the asymptotic running time of selection sort will be $\Omega(n^2)$ and will lie in of set of asymptotic function i.e $\Theta(n^2)$. Hence we can say that the asymptotic running time of old SS will be:

$$T(n) = On^2$$

### *D. Worst - Case Time Complexity.*

Now for the worst-case scenario when we have all $t_{ij} = 1$ so we have

$$T(n) = C_1 n + C_2(n-1) + C_3\frac{n(n-1)}{2} + C_3(n-1) + C_4\frac{n(n-1)}{2} + C_5\frac{n(n-1)}{2} + C_6(n-1) + C_7(n-1) + C_8(n-1)$$

$$T(n) = C_1 n + C_2 n - C_2 + \frac{C_3 n^2}{2} - \frac{C_3 n}{2} + C_3 n - C_3 + \frac{C_4 n^2}{2} - \frac{C_4 n}{2} + \frac{C_5 n^2}{2} - \frac{C_5 n}{2} + C_6 n - C_6 + C_7 n - C_7 + C_8 n - C_8$$

$$T(n) = \left(\frac{C_3}{2} + \frac{C_4}{2} + \frac{C_5}{2}\right) n^2 + \left(C_1 + C_2 + C_3 + C_6 + C_7 + C_8 - \frac{C_3}{2} - \frac{C_4}{2} - \frac{C_5}{2}\right) n - (C_2 + C_3 + C_7 + C_8)$$

(Eq: 01)

Let

$$a = \frac{C_3}{2} + \frac{C_4}{2} + \frac{C_5}{2}$$

$$b = C_1 + C_2 + C_3 + C_6 + C_7 + C_8 - \frac{C_3}{2} - \frac{C_4}{2}$$

$$c = -(C_2 + C_3 + C_7 + C_8)$$

If we suppose $C_1 = C_2 = C_4 = C_5 = C_6 = C_7 = C_8 = 1$ and put it in Equation 01
Then

$$T(n) = 3/2n^2 + 9/2\ n - 5$$

$$T(n) = an^2 + bn + c$$

Thus here in worst-case, the complexity of execution time of an algorithm shows the upper bound and is asymptotically denoted with Big-O. Therefore by ignoring the constant a, b,c and the lower terms of n, and taking only the dominant term i.e. $n^2$ , then the asymptotic running time of selection sort will be of the order of $O(n^2)$ and will lie in of set of asymptotic function i.e $\Theta(n^2)$. Hence we can say that the asymptotic running time of old SS will be:

$$T(n) = On^2$$

It means that the best and worst case asymptotic running time of selection sort is same i.e $T(n) = On^2$ however there may be little difference in actual running time, which will be very less and hence ignored.

## III. INSERTION SORT ALGORITHM

### A. Insertion Sort

Insertion sort algorithm is also one of oldest, easiest and most useful sorting algorithms for dealing with modicum of data set. If the first few objects are already sorted, an unsorted object can be inserted in the sorted set in proper place [10].

### B. Pseudo Code & Execution Time of Individual Statement of Insertion Sort

**Algorithm InsertionSort (X, n)** ▷ **X[0..n-1]**

| S.No | Iteration | Cost | Times |
|---|---|---|---|
| 1 | **FOR** j ← 2 **TO** length[**X**] | $C_1$ | $n$ |
| 2 | **DO** key ← *X*[*j*] | $C_2$ | $n-1$ |
| 3 | ▶ Put *X*[*j*] into the sorted sequence *X*[1 . . *j* − 1] | 0 | $n-1$ |
| 4 | *i* ← *j* − 1 | $C_4$ | $n-1$ |
| 5 | **WHILE** *i* > 0 and *X*[*i*] > key | $C_5$ | $\sum_{j=2}^{n} t_j$ |
| 6 | **DO** *X*[*i* +1] ← *X*[*i*] | $C_6$ | $\sum_{j=2}^{n} (t_j - 1)$ |
| 7 | *i* ← *i* − 1 | $C_7$ | $\sum_{j=2}^{n} (t_j - 1)$ |
| 8 | *X*[*i* + 1] ← key | $C_8$ | $n-1$ |

### C. Best-Case Time Complexity of Insertion Sort.

$$T(n) = C_1 n + C_2(n-1) + C_4(n-1) + C_5\sum_{j=2}^{n} t_j + C_6\sum_{j=2}^{n}(t_j - 1) + C_7\sum_{j=2}^{n}(t_j - 1) + C_8(n-1)$$

Equation – 01

If $t_j = 1$ for $j = 2, 3, ..., n$ and the best-case running time can be computed as follows:

$$T(n) = C_1 n + C_2(n-1) + C_4(n-1) + C_5\sum_{j=2}^{n}(1) + C_6\sum_{j=2}^{n}(1-1) + C_7\sum_{j=2}^{n}(1-1) + C_8(n-1)$$

$$T(n) = C_1 n + C_2(n-1) + C_4(n-1) + C_5(1) + C_6(1-1) + C_7(1-1) + C_8(n-1)$$

$$T(n) = (C_1 + C_2 + C_4 + C_5 + C_8)n - (C_2 + C_4 + C_8)$$

Suppose

$$a = (C_1 + C_2 + C_4 + C_5 + C_8)$$

And

$$b = -(C_2 + C_4 + C_8)$$

Then

$$T(n) = an + b$$

The punch line here is that the while-loop in line 5 executed only once for each *j*. This happens if given array *A* is already sorted.

$$T(n) = an + b = O(n)$$

It is a linear function of *n*.

### D. Worst-Case Time Complexity of Insertion Sort.

When $t_j = j$ for $j = 2, 3, ..., n$ then the worst-case running time can be computed from equation (1) as follows:

$$T(n) = C_1 n + C_2(n-1) + C_4(n-1) + C_5 \sum_{j=2}^{n}(j) + C_6 \sum_{j=2}^{n}(j-1) + C_7 \sum_{j=2}^{n}(j-1) + C_8(n-1)$$

$$T(n) = C_1 n + C_2(n-1) + C_4(n-1) + C_5[\frac{n(n-1)}{2} + 1] + C_6[\frac{n(n-1)}{2}] + C_7[\frac{n(n-1)}{2}] + C_8(n-1)$$

$$T(n) = [C_5/2 + C_6/2 + C_7/2]n^2 + [C_1 + C_2 + C_4 + C_5/2 - C_6/2 - C_7/2 + C_8]\ n - [(C_2 + C_4 + C_5 + C_8)] \quad \text{(Eq: 02)}$$

If we suppose $C_1 = C_2 = C_4 = C_5 = C_6 = C_7 = C_8 = 1$ and put it in Equation 02
Then

$$T(n) = [1/2 + 1/2 + 1/2]n^2 + [1 + 1 + 1 + 1/2 - 1/2 - \tfrac{1}{2} + 1]\ n - [1 + 1 + 1 + 1]$$

$$T(n) = 3/2n^2 + 7/2\, n - 4 \qquad \text{Eq: 03}$$

Suppose

$$a = [C_5/2 + C_6/2 + C_7/2]$$
$$b = [C_1 + C_2 + C_4 + C_5/2 - C_6/2 - C_7/2 + C_8]$$
$$c = -[(C_2 + C_4 + C_5 + C_8)]$$

Then

$$T(n) = an^2 + bn + c$$

This running time can be expressed as $an^2 + bn + c$ for constants *a*, *b*, and *c* that again depends on the statement costs $c_i$. Therefore, $T(n)$is a quadratic function of n.

We usually concentrate on finding the worst-case running time: the longest running time for any input size *n*.

## IV. OPTIMIZED SELECTION SORT ALGORITHM (OSSA)

### A. Optimized Selection Sort Algorithm

The three main characteristics of this algorithms is that, it is in-place, stable and easy to understand as compared to other algorithms of order $O(n^2)$ [15].

The working mechanism of OSSA is that it starts sorting the array from both ends. In a single iteration, first the smallest and largest element in the array are searched, and then both at the same time are placed at the specific location. This process is continued until the whole array is sorted [15].

Even though the order of OSSA is still $O(n^2)$, but its performance level has been very much improved as compared to other sorting algorithm of the said order i.e selection sort, insertion sort etc [15].

### B. Pseudo Code & Execution Time of Individual Statement of Optimized Selection Sort Algorithm

**Algorithm OSSA (X, n)** ▷ **X[0..n-1]**

| S# | Iteration | Cost | Times |
|---|---|---|---|
| 1 | k ← 0 | $C_1$ | 1 |
| 2 | for i ← n – 1 to k | $C_2$ | $\frac{n}{2} + 1$ |
| 3 | IndexOfLarge←IndexOfSmall←k | $C_3$ | $\frac{n}{2}$ |
| 4 | for j←k + 1 to i | $C_4$ | $\sum_{i=1}^{n/2} i + 1$ |
| 5 | if (X[j] > X(IndexOfLarge]) | $C_5$ | $\sum_{i=1}^{n/2} i$ |
| 6 | IndexOfLarge ← j | $C_6$ | $\sum_{i=1}^{n/2} t\, i_j$ |
| 7 | if (X[j] < X[IndexOfSmall]) | $C_7$ | $\sum_{i=1}^{n/2} i$ |
| 8 | IndexOfSmall← j | $C_8$ | $\sum_{i=1}^{n/2} t\, i_j$ |
| 9 | Large ← X[IndexOfLarge] | $C_9$ | $\frac{n}{2}$ |
| 10 | Small ← X[IndexOfSmall] | $C_{10}$ | $\frac{n}{2}$ |
| 11 | X[IndexOfLarge] ← X[i] | $C_{11}$ | $\frac{n}{2}$ |
| 12 | X[IndexOfSmall] ← X[k] | $C_{12}$ | $\frac{n}{2}$ |
| 13 | if (IndexOfLarg==k) Temp ← X[i] | $C_{13}$ | $\frac{n}{2}$ |

| | | | |
|---|---|---|---|
| | X[i] ← Large<br>X[IndexOfSmall] ← Temp | | |
| 14 | Else X[i] ← Large<br>Temp ← X[k]<br>X[k] ← Small<br>X[IndexOfLarge] ← Temp | $C_{14}$ | $\frac{n}{2}$ |
| 15 | Else X[k] ← Small<br>k ← k + 1 | $C_{15}$ | $\frac{n}{2}$ |

### C. *Best-Case Time Complexity of Optimized Selection Sort.*

$$T(n) = C_1 + C_1\frac{n}{2} + C_2\frac{n}{2} + C_2 + C_3\frac{n}{2} + C_4\sum_{i=1}^{\frac{n}{2}}(i+1) + C_5\sum_{i=1}^{\frac{n}{2}}(i) + C_6\sum_{i=1}^{\frac{n}{2}}(i) + C_7\sum_{i=1}^{\frac{n}{2}}(i) + C_8\sum_{i=1}^{\frac{n}{2}}(i) + C_9\frac{n}{2} + C_{10}\frac{n}{2} + C_{11}\frac{n}{2} + C_{12}\frac{n}{2} + C_{13}\frac{n}{2} + C_{14}\frac{n}{2} + C_{15}\frac{n}{2}$$

$$C_4\sum_{i=1}^{n/2}(i+1) = C_4\sum_{i=1}^{n/2}(i) + C_4\sum_{i=1}^{n/2}(1) = C_4\frac{\left(\frac{n}{2}+1\right)}{2} + C_4\frac{n}{2}$$

$$C_5\sum_{i=1}^{n/2}(i) = C_5\frac{\left(\frac{n}{2}+1\right)}{2} = C_5\frac{n^2}{8} + C_5\frac{n}{4}$$

$$C_6\sum_{i=1}^{n/2}(i) = C_6\frac{\left(\frac{n}{2}+1\right)}{2} = C_6\frac{n^2}{8} + C_6\frac{n}{4}$$

$$C_7\sum_{i=1}^{n/2}(i) = C_7\frac{\left(\frac{n}{2}+1\right)}{2} = C_7\frac{n^2}{8} + C_7\frac{n}{4}$$

$$C_8\sum_{i=1}^{n/2}(i) = C_8\frac{\left(\frac{n}{2}+1\right)}{2} = C_8\frac{n^2}{8} + C_8\frac{n}{4}$$

$$T(n) = C_1 + C_1\frac{n}{2} + C_2\frac{n}{2} + C_2 + C_3\frac{n}{2} + C_4\frac{\left(\frac{n}{2}+1\right)}{2} + C_4\frac{n}{2} + C_5\frac{n^2}{8} + C_5\frac{n}{4} + C_7\frac{n^2}{8} + C_7\frac{n}{4} + C_9\frac{n}{2} + C_{10}\frac{n}{2} + C_{11}\frac{n}{2} + C_{12}\frac{n}{2} + C_{13}\frac{n}{2} + C_{14}\frac{n}{2} + C_{15}\frac{n}{2}$$

$$T(n) = C_1 + C_1\frac{n}{2} + C_2\frac{n}{2} + C_2 + C_3\frac{n}{2} + C_4\frac{n^2}{8} + C_4\frac{n}{4} + C_4\frac{n}{2} + C_5\frac{n^2}{8} + C_5\frac{n}{4} + C_7\frac{n^2}{8} + C_7\frac{n}{4} + C_9\frac{n}{2} + C_{10}\frac{n}{2} + C_{11}\frac{n}{2} + C_{12}\frac{n}{2} + C_{13}\frac{n}{2} + C_{14}\frac{n}{2} + C_{15}\frac{n}{2}$$

$$T(n) = \left(\frac{C_5}{8} + \frac{C_4}{8} + \frac{C_7}{8}\right)n^2 + \left(\frac{C_1}{2} + \frac{C_2}{2} + \frac{C_3}{2} + \frac{C_4}{2} + \frac{C_4}{4} + \frac{C_5}{4} + \frac{C_7}{4} + \frac{C_9}{2} + \frac{C_{10}}{2} + \frac{C_{11}}{2} + \frac{C_{12}}{2} + \frac{C_{13}}{2} + \frac{C_{14}}{2} + \frac{C_{15}}{2}\right)n + (C_1 + C_2)$$

Let

$$a = \frac{C_5}{8} + \frac{C_4}{8} + \frac{C_7}{8}$$

$$b = \frac{C_1}{2} + \frac{C_2}{2} + \frac{C_3}{2} + \frac{C_4}{2} + \frac{C_4}{4} + \frac{C_5}{4} + \frac{C_7}{4} + \frac{C_9}{2} + \frac{C_{10}}{2} + \frac{C_{11}}{2} + \frac{C_{12}}{2} + \frac{C_{13}}{2} + \frac{C_{14}}{2} + \frac{C_{15}}{2}$$

$$c = (C_1 + C_2)$$

$$T(n) = an^2 + bn + c$$

### A. *Worst-Case Time Complexity of Optimized Selection Sort.*

$$T(n) = C_1 + C_1\frac{n}{2} + C_2\frac{n}{2} + C_2 + C_3\frac{n}{2} + C_4\sum_{i=1}^{\frac{n}{2}}(i+1) + C_5\sum_{i=1}^{\frac{n}{2}}(i) + C_6\sum_{i=1}^{\frac{n}{2}}(i) + C_7\sum_{i=1}^{\frac{n}{2}}(i) + C_8\sum_{i=1}^{\frac{n}{2}}(i) + C_9\frac{n}{2} + C_{10}\frac{n}{2} + C_{11}\frac{n}{2} + C_{12}\frac{n}{2} + C_{13}\frac{n}{2} + C_{14}\frac{n}{2} + C_{15}\frac{n}{2}$$

$$T(n) = C_1 + C_1\frac{n}{2} + C_2\frac{n}{2} + C_2 + C_3\frac{n}{2} + C_4\frac{n^2}{8} + C_4\frac{n}{4} + C_4\frac{n}{2} + C_5\frac{n^2}{8} + C_5\frac{n}{4} + C_6\frac{n^2}{8} + C_6\frac{n}{4} + C_7\frac{n^2}{8} + C_7\frac{n}{4} + C_8\frac{n^2}{8} + C_8\frac{n}{4} + C_9\frac{n}{2} + C_{10}\frac{n}{2} + C_{11}\frac{n}{2} + C_{12}\frac{n}{2} + C_{13}\frac{n}{2} + C_{14}\frac{n}{2} + C_{15}\frac{n}{2}$$

$$T(n) = \left(\frac{C_4}{8} + \frac{C_5}{8} + \frac{C_6}{8} + \frac{C_7}{8} + \frac{C_8}{8}\right)n^2 + \left(\frac{C_1}{2} + \frac{C_2}{2} + \frac{C_3}{2} + \frac{C_4}{2} + \frac{C_4}{4} + \frac{C_5}{4} + \frac{C_6}{4} + \frac{C_7}{4} + \frac{C_8}{4} + \frac{C_9}{2} + \frac{C_{10}}{2} + \frac{C_{11}}{2} + \frac{C_{12}}{2} + \frac{C_{13}}{2} + \frac{C_{14}}{2} + \frac{C_{15}}{2}\right)n + (C_1 + C_2)$$

If we put values of all C's equal to one then we get

$$T(n) = \frac{5}{8}n^2 + \frac{25}{4}n + 2$$

$$a = \frac{C_5}{8} + \frac{C_4}{8} + \frac{C_7}{8} + \frac{C_8}{8}$$

$$b = \frac{C_1}{2} + \frac{C_2}{2} + \frac{C_3}{2} + \frac{C_4}{2} + \frac{C_4}{4} + \frac{C_5}{4} + \frac{C_6}{4} + \frac{C_7}{4} + \frac{C_8}{4} + \frac{C_9}{2} + \frac{C_{10}}{2} + \frac{C_{11}}{2} + \frac{C_{12}}{2} + \frac{C_{13}}{2} + \frac{C_{14}}{2} + \frac{C_{15}}{2}$$

$$c = (C_1 + C_2)$$

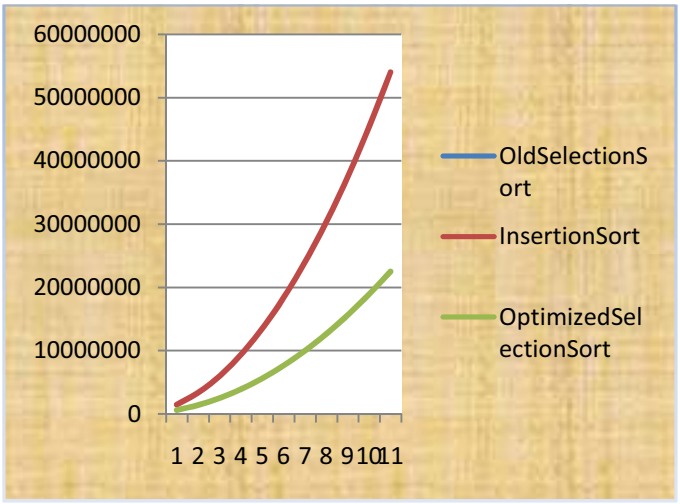
$$T(n) = an^2 + bn + c$$

## V. COMPARISON OF SELECTION SORT, INSERTION SORT AND OPTIMIZED SELECTION SORT ALGORITHM

Below is the table representing the calculated time, when multiple values of n are used.

| n | Calculated Time of SSA | Calculated Time of Optimized SSA | Calculated Time of Insertion Sort |
|---|---|---|---|
| n | T(n) Old SS | T(n) OSSA | T(n) InserT |
| 1000 | 1504495 | 631254 | 1503496 |
| 1500 | 3381745 | 1415629 | 3380246 |
| 2000 | 6008995 | 2512504 | 6006996 |
| 2500 | 9386245 | 3921879 | 9383746 |
| 3000 | 13513495 | 5643754 | 13510496 |
| 3500 | 18390745 | 7678129 | 18387246 |
| 4000 | 24017995 | 10025004 | 24013996 |
| 4500 | 30395245 | 12684379 | 30390746 |
| 5000 | 37522495 | 15656254 | 37517496 |
| 5500 | 45399745 | 18940629 | 45394246 |

Below is the line graph plotted in response of data given in above table.




The graph shows that optimized selection is far better than its competitor algorithms of the same order of O ($n^2$).

## Conclusion

OSSA is basically based on the logic of old Selection Sort, but it perform less amount of iteration that's why it save almost fifty percent of the execution time. OSSA is a new comparison based sorting algorithm, with an $O(n^2)$ order complexity, yet it is the fastest of its kinds, like selection and insertion sort in the worst case. The results proved the claim.

## REFERENCES

[1] Flores, I. "Analysis of Internal Computer Sorting". J.ACM 7,4 (Oct. 1960), 389- 409.

[2 ] G. Franceschini and V. Geffert. "An In-Place Sorting with O (n log n) Comparisons and O (n) Moves". In Proc. 44th Annual IEEE Symposium on Foundations of Computer Science, pages 242-250, 2003.

[3] Knuth, D."The Art of Computer programming Sorting and Searching", 2nd edition, vol.3. Addison- Wesley, 1998.

[4] Hoare, C.A.R. "Algorithm 64: Quick sort". Comm. ACM 4,7 (July 1961), 321.

[5] Soubhik Chakraborty, Mausumi Bose, and Kumar Sushant, A Research thesis, On Why Parameters of Input Distributions Need be Taken Into Account For a More Precise Evaluation of Complexity for Certain Algorithms.

[6] D.S. Malik, C++ Programming: Program Design Including Data Structures, Course Technology(Thomson Learning), 2002, www.course.com.

[7] D. Jim´enez-Gonz´alez, J. Navarro, and J. Larriba-Pey. CC-Radix: "A Cache Conscious Sorting Based on Radix Sort". In Euromicro Conference on Parallel Distributed and Network based Processing, pages 101-108, February 2003.

[8] J. L. Bentley and R. Sedgewick. "Fast Algorithms for Sorting and Searching Strings", ACM-SIAM SODA "97, 360-369, 1997.

[9] Flores, I."Analysis of Internal Computer Sorting". J.ACM 8, (Jan. 1961), 41-80.

[10] Williams, J.W.J. "Algorithm 232: Heap sort". Comm. ACM 7, 6 (June 1964), 347-348.

[11] ANDERSSON, A. and NILSSON, S. 1994. "A New Efficient Radix Sort". In the Proceeding of the 35 Annual IEEE Symposium on Foundation of Computer Science (1994),pp. 714-721.

[12] DAVIS, I.J. 1992. "A Fast Radix Sort". The computer journal 35, 6, 636-642.

[13] V.Estivill-Castro and D.Wood."A Survey of Adaptive Sorting Algorithms", Computing Surveys, 24:441-476, 1992.

[14] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein."Introduction to Algorithms". MIT Press, Cambridge, MA, 2nd edition, 2001.

[15] Sultanullah Jadoon, Salman Faiz, Salim ur Rehman, Hamid Jan "Design & Analysis of Optimized Selection Sort Algorithm", IJEC-IJENS Volume 11 Issue 01, 2011.